Name(last, first): Cheung, Vincent Midterm 20 10/30/2018

1. (20 points) Consider a set of intervals $I_1, I_2, \cdots, I_n$:

   (a) Design a linear time algorithm (assume that intervals are sorted in any manner you wish) that assigns the intervals to the minimum number of processors.

   (b) Prove the correctness of your algorithm.

a) Assume the intervals are sorted from earliest to latest start time.
Assume we also possess the minimum number of processors for these intervals.

Algorithm: For each interval,
- Assign the interval to any processor that is available when the interval starts

This is $O(n)$ because there is constant work for each interval, and each interval is visited once.

b) Proof. Let the density be the greatest number of overlapping intervals for any time.

In the algorithm, we know that there will be an available processor for each interval at each iteration. Suppose interval I_i could not find an available processor at its start time. Then there must have been density number of intervals currently running on density number of processors. But I_i overlaps with these intervals at I_i's start time, so there are density + 1 number of overlapping intervals. But density is the greatest number of overlapping intervals, a contradiction. So there must be an available processor for I_i, and so we have assigned every interval to the minimum number (density) of processors.

→ would've been better to describe what part of algorithm and how compute the density

16/20

Name(last, first): Cheung, Vincent   Midterm   10/30/2018

2. (20 points) (a) Design an efficient algorithm that outputs the vertices of a DAG (Directed Acylic Graph), such that if there is an edge (x, y) then x is output before y.

(b) Analyze the run time of your algorithm.

a)
- Find a vertex with no incoming edges (a source).
- Output this vertex and remove it and its outgoing edges from the graph.
- Repeat the above steps until no more vertices remain.

10/10

Proof. We can always find a source at each iteration of the algorithm. Suppose we could not. Then each vertex would have at least one incoming edge. If we go backwards along these incoming edges repeatedly more than n times, we will eventually repeat vertices, which means there is a cycle — a contradiction, since DAGs have no cycles

Moreover, we know x is output before y if there is an edge (x,y). Suppose that y were output before x. Then y would not be a source since the incoming edge (x,y) exists, so our algorithm would not output y before x.

b)
- We first run through the graph and count the number of incoming edges ("indegree") of each vertex. This is $O(m+n)$. ✓
- A source is now any vertex with indegree 0. When we remove a source, we look to its outgoing edges and reduce the indegree of the adjacent vertices. This is also $O(m+n)$.

⇒ Overall, the time complexity is $O(m+n)$.

missing some breakdown

6/10

15

3. (20 points) An undirected graph is said to have property X if you can start from a vertex, traverse all edges of the graph exactly once, without removing your pen from the paper.
   (a) Classify the graphs that have property X?
   (b) Design an efficient algorithm for generating a traversal of a graph that has property X.

a) These are connected graphs that have two or fewer nodes with odd degree. If a node has odd degree, it must occur at either the start or end of the traversal, so there can be at most two. ✓

b)
- Pick an odd-degree node with the lowest degree. If none, pick any node
- Pick an unused edge to a node that is incident on more than one unused edge, and traverse it

-5

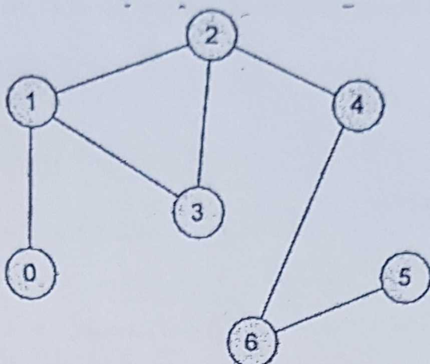
- Repeat until there is only one node incident on just one unused edge. Traverse this edge.

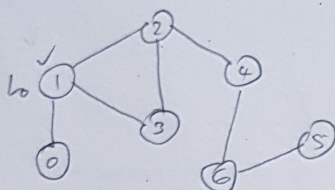
Proof. We are guaranteed to start on an odd degree node, if there are any, by step 1 of the algorithm. We can always find an unused edge in step 2 since the graph is connected, and the criterion for picking the node is that it is incident on more than one unused edge. Finally, we end on a node of odd degree — we know it is odd since this node must have been "entered" and "exited" an even number of times, plus this final "entrance", so it must have had an odd degree.

10

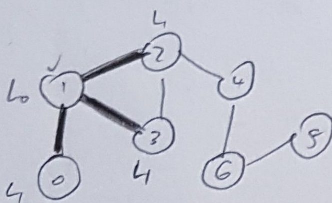
4. (10 points) Consider an unweighted graph G shown below:

   (a) Starting from vertex 1, show every step of BFS along with the corresponding FIFO next to it.

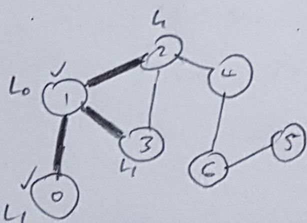




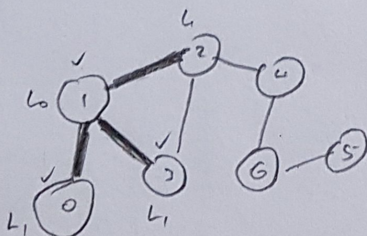
FIFO
1



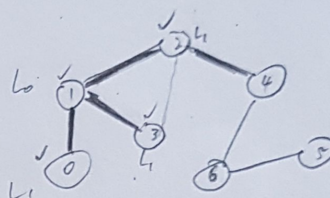
FIFO
0
3
2



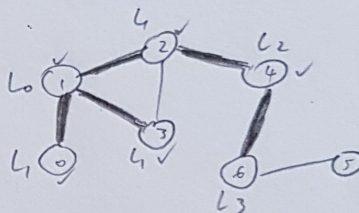
FIFO
3
2



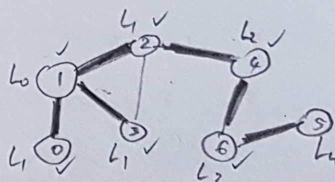
FIFO
2



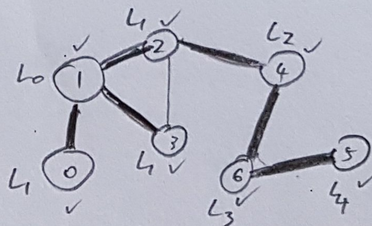
FIFO
4



FIFO
6

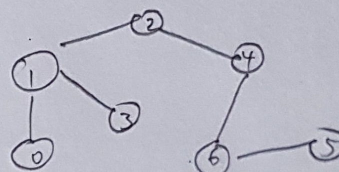


FIFO
5



FIFO
(empty)

RESULT:



5. (20 points) Consider an unsorted list of integers. You can find the minimum number in the list with $n-1$ comparisons. Similarly, you can find the maximum with $n-1$ comparisons. So you can find both the minimum and the maximum with about $2n-3$ comparisons. Design an algorithm that finds both the minimum and the maximum using about $\frac{3n}{2}$ comparisons.

- Initialize max = 0, min = 0
- Scan through the list.
  - If the current element > max,
    - Set max = current element
    - Continue immediately to the next element
  - If the current element < min
    - Set min = current element.

$2n$

Proof. Suppose there was an element greater than the max. Our algorithm guarantees that max will be set to this element, so we will always find the max. The proof is similar for the min.

\# comparisons. The first comparison will always occur for all n elements. The second comparison will only occur if the first one returned false. The first comparison will return false on average half the time, so the second comparison will occur $\frac{n}{2}$ times.

$\Rightarrow$ Overall, this algorithm uses $\frac{3n}{2}$ comparisons.

6. (10 points) Give an algorithm to color a graph with 2 colors (assuming it is 2-colorable). A proof of correctness is not necessary.

- Run BFS, but with these modifications:
  - If a node is on an even layer, give it one color.
  - Else if a node is on an odd layer, give it the other color.

-4 Time complexity

OK